# A gráfelmélet néhány alkalmazása

A gráfelmélet néhány szemléletes és jelents alkalmazására adunk példát ebben a fejezetben. A fagráf bejárása a különböz mveleti kódok alapját képezi. A minimális feszítfa megkeresésére vonatkozó számos algoritmus közül a Kruskal-algoritmussal foglalkozunk, míg a legrövidebb út problémáját Dijkstra algoritmusának bemutatásával tárgyaljuk.

## Fagráf bejárása

### Eljárások

```
> restart;
> with(GraphTheory):
> kisebb_mint:=proc(a,b)
  local t;
    if type([a,b],[string,string]) then
      t:=sort([a,b],lexorder);
    else
      t:=sort([a,b])
   fi;
   if a=t[1] then true else false fi;
  end:
> Preorder:=proc(G::Graph,r)
   local Dep,v,T;
   #A gyökér meglátogatása
   printf("%a",r);
   #A gyökér gyerekeinek meghat.
   Dep:=Departures(G,r):
   #A gyerekek sorbarendezése
   Dep:=sort(convert(Dep,list),kisebb_mint);
   #A részfák Preorder bejárása
   for v in Dep do
    Preorder(G,v)
   od; printf("\n",r)
  end:
> Inorder:=proc(G::Graph,r)
  local v,Dep,T;
  #Ha levélhez értünk el, akkor kiírjuk a csúcs nevét
  if OutDegree(G,r)=0 then
   print(r):
  #Bels csúcsnál járunk
```

```
  else
   Dep:=Departures(G,r);
   # Meghatározzuk a gyerekek sorrendjét
   # és bejárjuk a legbaloldalibb részfát
   Dep:=sort(convert(Dep,list),kisebb_mint);
   Inorder(G,Dep[1]);
   #Meglátogatjuk a gyökeret
   print(r);
   # Bejárjuk inorder módon a visszalev részfákat
   for v in Dep[2..nops(Dep)] do
    Inorder(G,v)
   od;
  fi;
  end:
> Postorder:=proc(G::Graph,r)
  local v,Dep,T;
  #Tekintjük a györér gyerekeit
   Dep:=Departures(G,r);
   # Sorba rendezzük a gyerekeket
   Dep:=sort(convert(Dep,list),kisebb_mint);
   # Rendre bejárjuk részfákat postorder módon
   for v in Dep do
    Postorder(G,v)
   od;
  #Meglátogatjuk a gyökeret
  printf(" %c",r)
  end:
> alakit:=proc(G::Graph)
  local L,L1;
  L:=map(t->convert(t,list),Edges(G));
  L1:=map(t->sort(t,kisebb_mint),L);
  Digraph(L1);
  end:
```

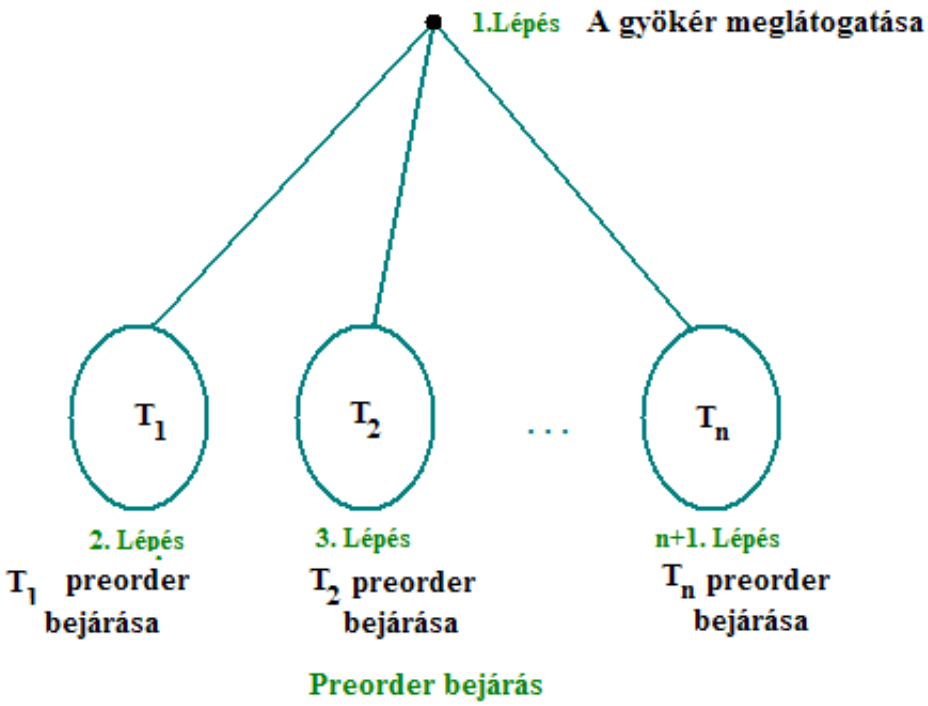
A fákkal kapcsolatos algoritmusoknál sok esetben végig kell járni a csúcsokat. Ilyenkor a fa bejárásáról beszélünk. Mivel egy n csúcsú fa alakja elég változatos lehet, egy ilyen algoritmus felépítése nem kézenfekv. Három alapvet eljárást mutatunk a fa bejárására. Az ültetett fát részfák rendszerének fogjuk fel, s rekurzív módon járjuk be a fát. A különböz típusú fa-bejárásokat a következ gráf bejárásával szemléltetjük:

```
> G:= Graph([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p]):
  AddEdge(G,{{a,b},{a,c},{a,d},{b,e},{b,f},{d,g},{d,h},{d,i},{e,
  j},{e,k},{g,l},{g,m},{k,n},{k,o},{k,p}});
```

```
> DrawGraph(G,style=tree,root=a);
```

## Preorder bejárás

A preorder bejárás esetén
Elször megvizsgáljuk a gyökeret,
ezután rendre bejárjuk balról jobbra  a részfákat




A preorder bejárást valósítja hasonnev, **Preorder** eljárás valósítja meg. Paraméterei a gráf és a gyökér. Rajzoljuk fel mindenekeltt a fagráfot.

```
> DrawGraph(G,style=tree,root=a);
```

A gráf bejárásához át kell alakítanunk a gráfot irányított gráffá, amelynek élei mindig a kisebb cimkéj csúcsból a nagyobb cimkéj csúcsba mutatnak.  Az átalakítást az **alakit**  eljárással végezzük:

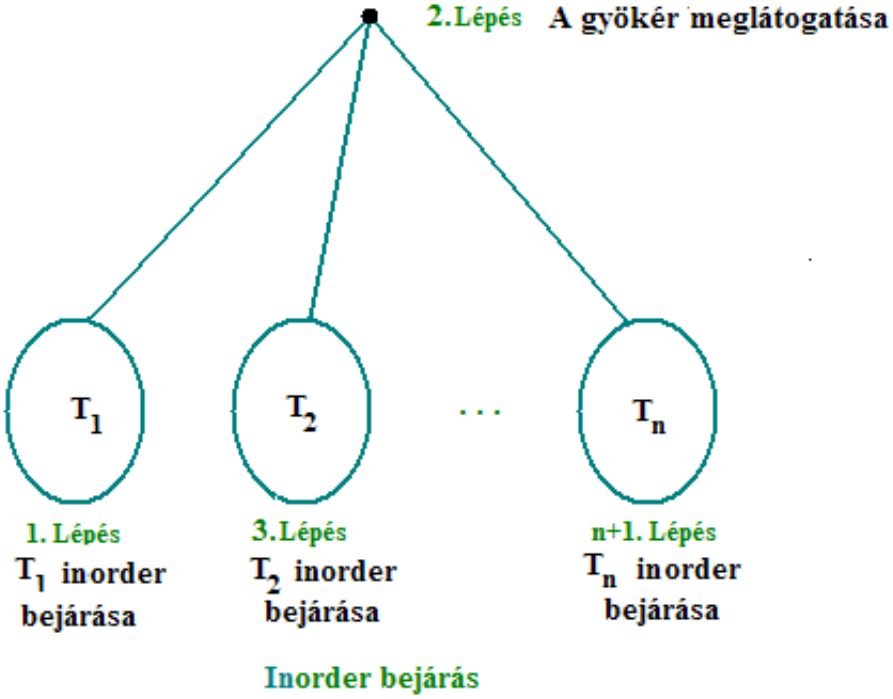
```
> Gd:=alakit(G);
```

Ezután alkalmazhatjuk a Preorder eljárást.

```
> Preorder(Gd,a);
```

## Inorder bejárás

Inorder bejárás esetén
Elször bejárjuk a bal széls részfát,
majd megvizsgáljuk a gyökeret,
ezután balról jobbra bejárjuk a többi részfát.

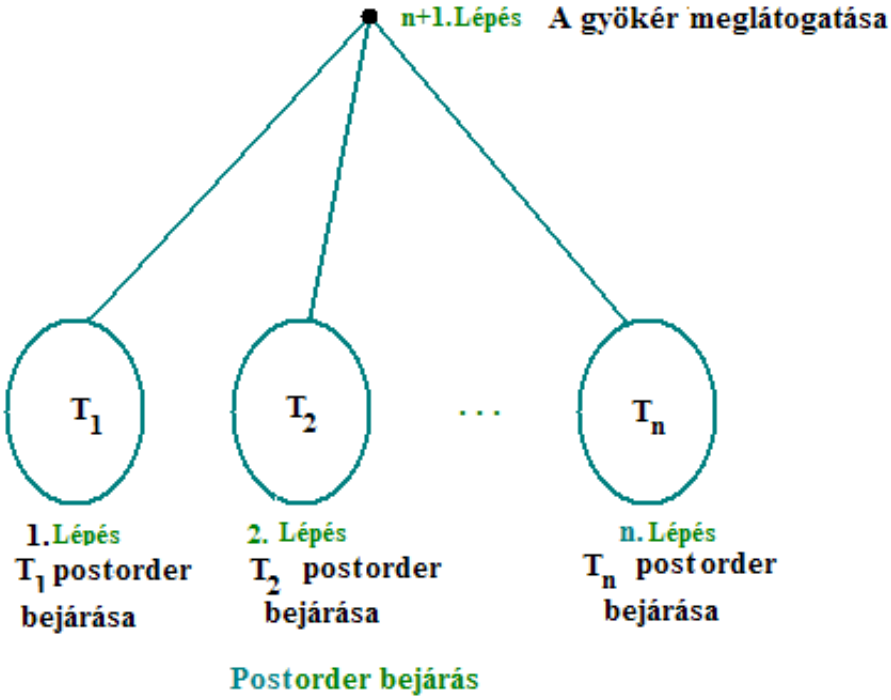



Az alábbi eljárás a inorder bejárást valósítja meg. Paraméterei a gráf és a gyökér.

```
> DrawGraph(G,style=tree,root=a),Inorder(Gd,a);
```

## Postorder bejárás

Postorder bejárás esetén
Bejárjuk rendre balról jobbra a részfákat,
megvizsgáljuk a gyökeret




Az alábbi eljárás a postorder bejárást valósítja meg. Paraméterei a gráf és a gyökér.

```
> DrawGraph(G,style=tree,root=a),Postorder(Gd,a);
```

Természetesen kisebb méret gráfok esetén a különböz bejárások az eljárások használat nélkül, közvetlenül is elvégezhetk. Tekintsük például a következ, G2-vel jelölt bináris gráfot:

```
> G2:= Graph([a,b,c,d,e,f,g,h,i,j,k]):
  AddEdge(G2,{{a,b},{a,c},{b,d},{b,e},{c,f},{c,g},{d,h},{d,i},
  {f,j},{f,k}});
> DrawGraph(G2,style=tree,root=a);
```

Járjuk be a gráfot pédául postorder módon. Az eredmény: h, i, d, e, b, j, k, f, g, c, a.
Ellenrizzük eredményünket a Postorder eljárással:

```
> G2d:=alakit(G2);Postorder(G2d,a);
>
```

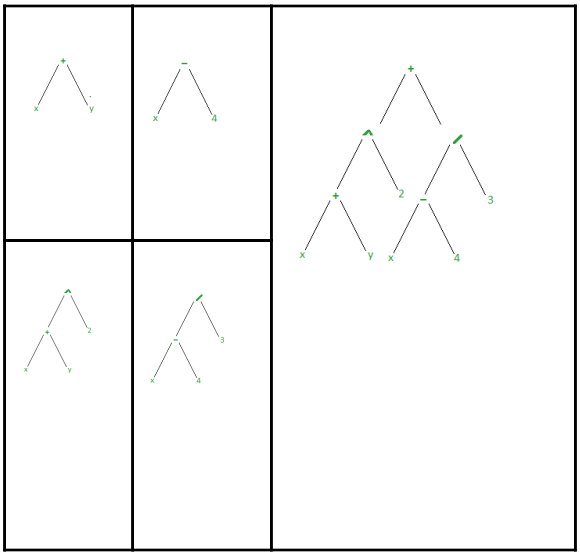
## Mveleti kódok

Tekintsük a következ kifejezést

**((x +y )^2) + ((x - 4) / 3)**

Ez a kifejezés un. infix mveleti kódolással van felírva, vagyis az operandusok közrefogják a mveleti jelet. Alkossuk meg a **mvelethez tartozó rendezett bináris fát**! A fát alulról fölfelé építjük fel. Elször elkészítjük az x+y kifejezésnek megfelel részfát, s ezt beágyazzuk az (x+y) ^2 kifejezést reprezentáló részfába. Hasonlóan elkészítjük az (x-4) -et megjelenít részfát, majd az

(x-4)/3-nak megfelel részfát.Végül az (x+y)^2 és az (x-4)/3 kifejezésekbl, a gyökérbe a + mveleti jelet téve létrehozzuk az ((x+y)^2) + ((x-4)/3) kifejezést reprezentáló fát. Ezeket a lépéseket láthatjuk az 1. ábrán.



1. ábra

A mveletsort ábrázoló gráf inorder bejárása a mveletsor inorder kódját reprezentálja. A mveletsor un. **prefix kód**ját nyerhetjük, ha a gráfot (jobboldali ábra) preorder módon járjuk be. A prefix jelölés vagy prefix kód megkülönböztet jellemzje az, hogy az operátorokat (mveleti jeleket) az operandusoktól balra helyezi el. Elnye, hogy egyértelmsége miatt zárójelekre nincs szükség. Járjuk be tehát a gráfot preorder módon! Az elz paragrafusban ezt már megtettük közvetlenül, az ábrát szemlélve. A prefix kód :

$$+ \wedge + x\ y\ 2\ /\ -\ x\ 4\ 3.$$

Hasonlóképpen kaphatjuk a bináris fa alapján a **postfix kód**ot a fa postorder bejárásával. A kifejezés postfix alakjában a bináris operátor követi az operandusokat. Az egyértelmség miatt zárójelekre itt sincs szükség. A szóban forgó kifejezés postfix kódja:

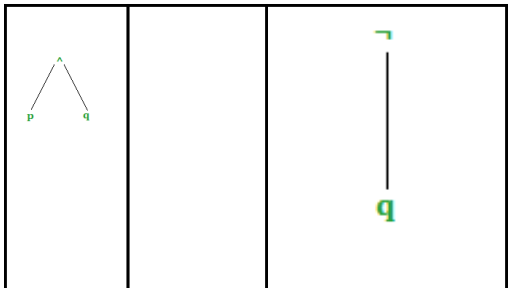
$$x\ y\ +\ 2\ \wedge\ x\ 4\ -\ 3\ /\ +.$$

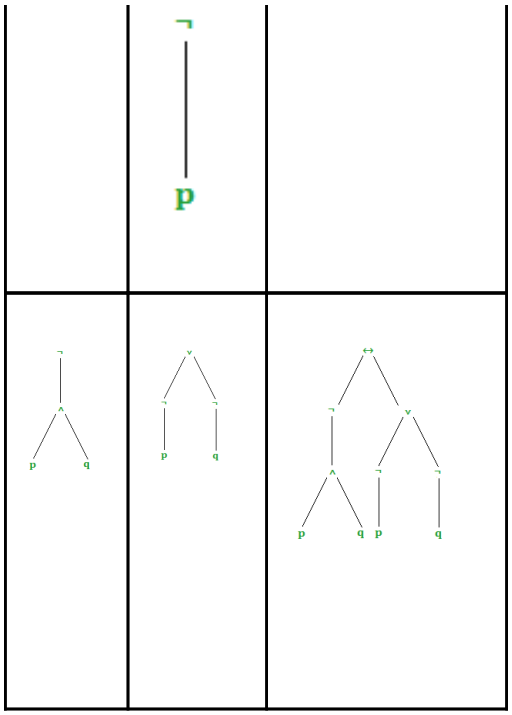
### 8.1.1. Példa

Szerkesszük meg a következ összetett logikai kifejezést reprezentáló rendezett, bináris, gyökeres fát. Ezután a fa segítségével írjuk föl a kifejezés prefix, infix és postfix alakját.

### Megoldás

A fát most is alulról fölfelé építjük fel. Elször a $\neg p$-nek és a $\neg q$-nak, valamint a $p \wedge q$-nak megfelelő részfát alakítjuk ki (itt a $\neg$ unáris operátor). Majd a $\neg (p \wedge q)$ és a $(\neg p) \vee (\neg q)$jön sorra. Végül az utóbbi két részfából megépítjük a végs gyökeres fát . (2. ábra)






2. ábra

A prefix, infix és postfix kódot megkapjuk, ha a kapott gyökeres fát bejárjuk rendre preorder, inorder illetve postorder módon.
A prefix kód: ↔ ¬ ∧ p q ∨ ¬ p ¬ q. Az infix kód: (¬(p∧q))↔((¬p)∨(¬q)). A postfix kód p q ∧ ¬ p ¬ q ∨ ↔.

A következkben példákat mutatunk a prefix ill. postfix kódú kifejezések kiértékelésére.

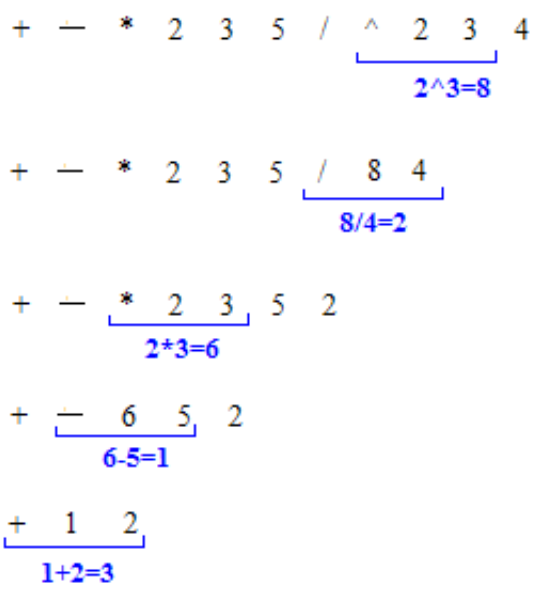
### 8.1.2. Példa

*Értékeljük ki a következ prefix kódú kifejezést:*

**+ − * 2 3 5 / ^ 2 3 4**

### Megoldás

A prefix alakú kifejezést jobbról balra haladva, a következk szerint értékeljük ki:
Megkeressük az els mveleti jelet (operátort), majd végrehajtjuk a mveletet a tle közvetlenül jobbra es két operandussal. A mvelet elvégzése után az eredményt új operandusként kezeljük, a az elz hármas (operátos és a két operandus) helyére illesztjük. A fenti lépéssort folytatjuk mindaddig, míg meg nem kapjuk a végeredményt.




A kifejezés értéke 3.

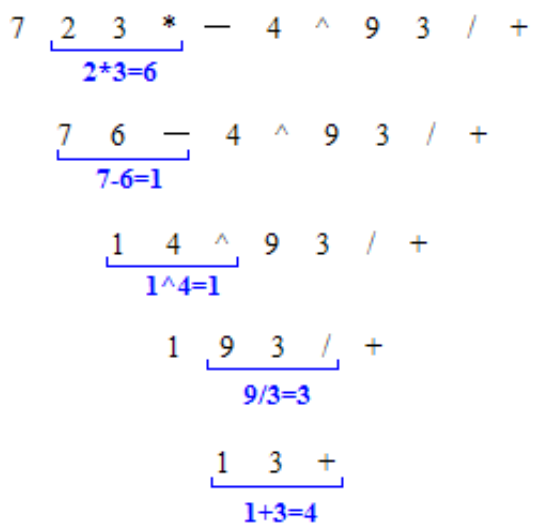
### 8.1.3. Példa

*Értékeljük ki a következ postfix kódú kifejezést:*

**7 2 3 * – 4 ^ 9 3 / +**

### Megoldás

A postfix alakú kifejezéstbalról jobbra haladva, a követezk szerint értékeljük ki: Megkeressük az els mveleti jelet (operátort), majd végrehajtjuk a mveletet a tle közvetlenül balra es két operandussal. A mvelet elvégzése után az eredményt új operandusként kezeljük, a az elz hármas (operátos és a két operandus) helyére illesztjük. A fenti lépéssort folytatjuk mindaddig, míg meg nem kapjuk a végeredményt.




A kifejezés értéke 4.

# Minimális feszítfa, Kruskal algoritmusa

## A minimális feszítfa fogalma

Legyen adott n számú település, amelyeket úthálózattal szeretnénk összekötni úgy, hogy bármely településbl el lehessen jutni a többibe. Ismerjük továbbá az egyes településeket összeköt utak építési költségeit. Keressük a legkisebb költség megoldást!

A gráfelmélet nyelvére lefordítva ez a következt jelenti: Adott egy összefügg, egyszer gráf, súlyozott élekkel, ezek a „költségek”. Keressünk egy minimális költségű (költségösszegű) olyan részgráfot, mely a gráf összes pontját tartalmazza!

Egy ilyen részgráfnak nyilván fagráfnak kell lennie, ugyanis ha volna benne kör, akkor annak egy tetszleges élét elhagyva - ezzel a költségek összegét csökkentve - összefügg maradna a gráf. A keresett gráf tehát egy olyan fa, mely a gráf összes pontját tartalmazza, azaz "feszít fa". Természetesen a megoldandó probléma sokféleképpen megfogalmazható. Szóba jöhetett volna például az is, hogy települések egy rendszerét vízvezeték-hálózatba akarjuk bekapcsolni, s rendre meghatározható a településeket összeköt vezetékrész építési költsége. Adott a vízforrás, melyik közvetlen összeköttetéseket építsük meg, hogyminden településre eljusson a víz, s a költség minimális legyen.

Adjuk meg pontosan a feszít fa fogalmát!

### 8.2.1. Definíció

Az összefügg G gráf F részgráfját G egy favázának (feszít fájának) nevezzük, ha F fagráf, és G minden csúcsát (pontját) tartalmazza.

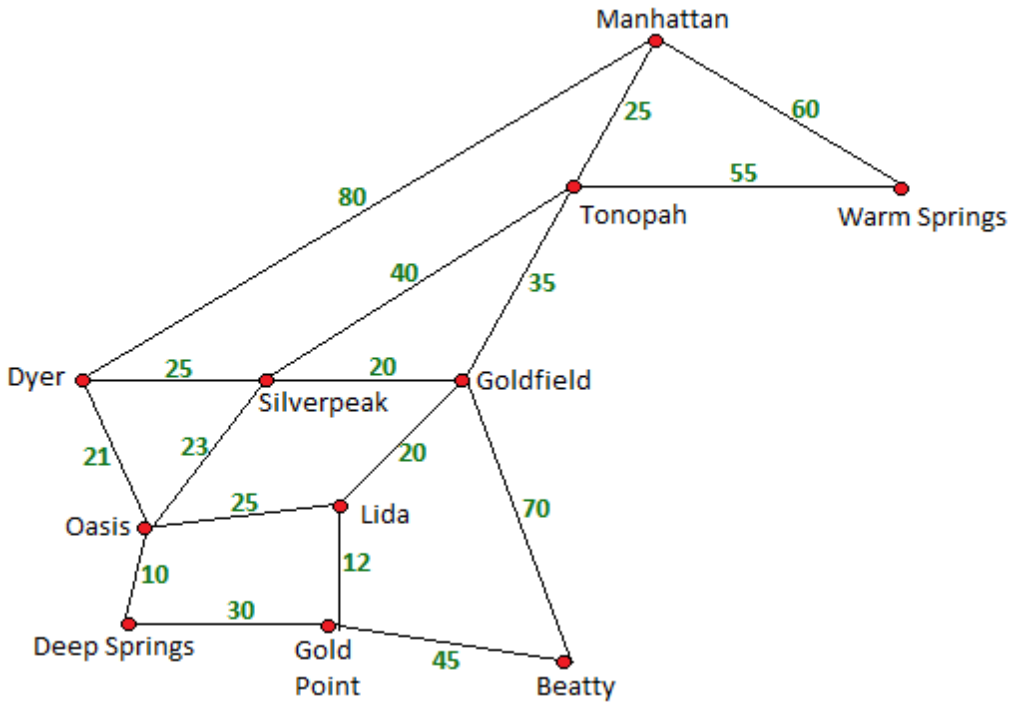
**8.2.2.Definíció**
Minimális érték faváz alatt az összefgg, súlyozott él gráf olyan favázát (feszít fáját) értjük, amelyre az élek súlyainak összege a lehet legkisebb.

A minimális feszítfa nem feltétlenül egyértelm, de annak súlya igen. A következkben a minimális feszítfa meghatározását Joseph Bernard Kruskal (született 1928-ban) másodéves egyetemi hallgató korában megalkotott algoritmusával végezzük.

## Probléma

Nevada államban sok, a képen láthatóhoz hasonló, burkolatlan út van.

Az alábbi gráf ilyen utak hálózatát ábrázolja. Az élek súlya a településpárok közti útszakaszok hosszát jelenti. Melyik útszakaszokat kellene burkolattal ellátni, hogy bármely településbl bármely településbe eljuthassunk burkolt úton és az útburkolás költsége a legkisebb legyen, vagyis a lehet legrövidebb úthosszat kelljen burkolni?




Hozzuk létre a problémához tartozó gráfot!

```
> restart:
> with(GraphTheory):
> G := Graph({[{Manhattan,Warm_S},60],[{Manhattan,Dyer},80],[
  {Manhattan,Tonopah},25],[{Warm_S,Tonopah},55],[{Tonopah,
  Goldfield},35],[{Tonopah,Silverpeak},40],[{Dyer,Silverpeak},
  25],[{Silverpeak,Goldfield},20],[{Dyer,Oasis},21],[
  {Silverpeak,Oasis},23],[{Goldfield,Lida},20],[{Goldfield,
  Beatty},70],[{Lida,Gold_P},12],[{Deep_S,Gold_P},30],[{Beatty,
  Gold_P},45],[{Oasis,Deep_S},10],[{Oasis,Lida},25]});
> DrawGraph(G);
```

A következkben a minimális feszítfa meghatározását Joseph Bernard Kruskal (1928-2010) másodéves egyetemi hallgató korában megalkotott algoritmusával végezzük. A Kruskal-algoritmus egy súlyozott gráfokat feldolgozó mohó algoritmus. A mohó algoritmus vagy greedy algoritmus olyan problémamegoldó algoritmus, amely helyi optimumok megvalósításával próbálja megtalálni a globális optimumot.

Ha a gráf összefügg, akkor a Kruskal algoritmus minimális feszítfa megalkotására szolgál, ha nem, akkor minimális feszít erdt hoz létre.

## A Kruskal algoritmus pszeudokódja

Az algoritmus lépései a következek:

- Válasszunk ki egyminimális súlyú élt.
- Ha ennek az élnek a részgráfhoz való hozzáadásával kör keletkeznék, "dobjuk azt el", egyébként vegyük hozzá a gráfhoz.
- Ha van még meg nem vizsgált él, folytassuk az elz lépésekkel.

Mindezt pontosabban mutatja be az algorimus pszeudokódja.
**procedure Kruskal**(G: súlyozott, összefügg, irányítatlan gráf n csúccsal)
T: = egy minimális súlyú él
**for** i:=1 **to** n-2 do
**begin**
e: = egy minimális súlyú él, amely nem hoz létre kört, ha hozzávesszük T-hez
T: = T -hez hozzáadva e-t
**end** { T egy minimális feszítfa G-ben }
Az algoritmus részletes kidolgozásával itt nem foglalkozunk, de bemutatjuk a Maple GraphTheory csomagjának Kruskal algoritmusát megvalósító eljárását.

## A Maple eljárása Kruskal algoritmusára

Oljuk meg az elz szekcióban megfogalmazott problémát!

**Megoldás**

Rajzoljuk fel újra a gráfot

```
> DrawGraph(G);
```

Ha valamely él súlya nem olvasható le egyértelmen az ábráról, akkor a következ utasítással kaphatjuk meg az értéket:

```
> GetEdgeWeight(G,{Dyer, Manhattan});
```

Ha az összes él súlyát szeretnénk tudni, akkor a következképpen járhatunk el:

```
> map(t->[t,GetEdgeWeight(G,t)],Edges(G));
```

...vagy a Maple beépített eljárását használva

```
> Edges(G,weights);
```

Ha az algoritmus futását leíró üzeneteket is akarunk kapni, akkor az információszintet szabályozó utasítást kell adnunk:

```
> infolevel[KruskalsAlgorithm] := 4:
```

A KruskalsAlgorithm els paramétere a gráf, a második, opcionális, paraméter használata esetén megkapjuk a minimális feszítfa összsúlyát is.

```
> T := KruskalsAlgorithm(G,'w'):
> `A minimális feszítfa élsúlyainak összege`=w;
```

A minimális feszítfa élsúlyainak összegét megkaphatjuk a GetEdgeWeight és az add utasítások együttes alkalmazásával is:

```
> add(GetEdgeWeight(G,e), e=Edges(T));
> `A minimális feszítfa élei a súlyukkal`= map(t->[t,
  GetEdgeWeight(T,t)],Edges(T));
```

Ugyanez a Maple beépített eljárásával:

```
> Edges(T,weights);
```

A következ utasítással a megtalált minimális feszítfát kitüntetett színnel belerajzoljuk az eredeti gráfba

```
> HighlightSubgraph(G,T,red,green);
```

```
> DrawGraph(G);
```

Rajzoljuk föl csak a gazdaságos, minimális összsúlyú feszítfát! Az ábrán látható fagráf azokat az útszakaszokat tartalmzza, amelyek megépítése esetén minden településbl minden településbe eljuthatunk burkolt úton, s az útszakaszok összhossza minimális, 266 méföld.

```
> DrawGraph(T);
```

### 8.2.3. Példa

*Hozzunk létre, véletlenszeren generált, összefügg, súlyozott él, egyszer gráfot, és határozzuk meg a gráf minimális feszítfáját!*

**Megoldás**

A RandomGraphs csomag Randomgraph utasítását használjuk a gráf generálásához, ezértbetöltjük acsomagot.

```
> with(RandomGraphs):
```

Az eljárás els paramétere a gráf csúcspontjainak száma, a második annak p valószínsége (0<= p<=1), hogy adott él szerepel a gráfban, a harmadik paraméter az élek lehetséges súlyának intervallumát adja meg, míg a connected opció jelentése "összefügg" . Bár az elz példánál leírtuk az egyes lépések magyarázatát, ezt most újra megtesszük, az ismert latin mondásra gondolva: "Repeticio est mater studiorum", magyarul: Ismétlés a tudás anyja.

```
> G := RandomGraph(10, 0.4, weights=1 .. 12, connected):
> DrawGraph(G);
```

Ha valamely él súlya nem olvasható le egyértelmen az ábráról, akkor a következ utasítással kaphatjuk meg az értéket:

```
> GetEdgeWeight(G,{1,10});
```

Ha az összes él súlyát szeretnénk tudni, akkor a következképpen járhatunk el:

```
> map(t->[t,GetEdgeWeight(G,t)],Edges(G));
```

...vagy a Maple beépített eljárását használva

```
> Edges(G,weights);
```

Ha az algoritmus futását leíró üzeneteket is akarunk kapni, akkor az információszintet szabályozó utasítást kell adnunk:

```
> infolevel[KruskalsAlgorithm] := 4:
> T := KruskalsAlgorithm(G,'w'):
> `A minimális feszítfa élsúlyainak összege`=w;
```

A minimális feszítfa élsúlyainak összegét megkaphatjuk a GetEdgeWeight és az add utasítások együttes alkalmazásával is:

```
> add(GetEdgeWeight(G,e), e=Edges(T));
> `A minimális feszítfa élei a súlyukkal`= map(t->[t,
  GetEdgeWeight(T,t)],Edges(T));
```

Ugyanez a Maple beépített eljárásával:

```
> Edges(T,weights);
```

A következ utasítással a megtalált minimális feszítfát kitüntetett színnel belerajzoljuk az eredeti gráfba

```
> HighlightSubgraph(G,T,red,green);
```

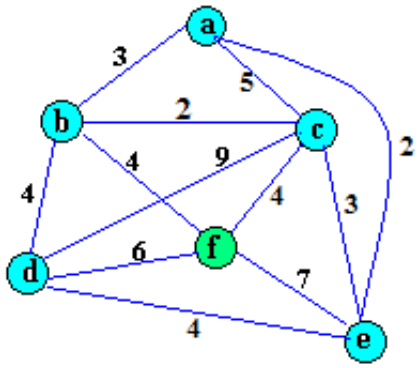
```
> DrawGraph(G);
```

Rajzoljuk föl csak a gazdaságos feszítfát:

```
> DrawGraph(T);
```

### 8.2.4. Példa

*Adjuk meg az alábbi gráf egy gazdaságos ( tehát minimális élköltség ) favázát! Mennyi a minimális összköltség?*




### Megoldás

Adjuk meg elször is a gráfot. Az éleket a súllyal együtt adjuk meg. Az [{a,b},s] formátum jelentése: a és b az él végpontjai, az s pedig az illet él súlya. Mindezeket listába folgaljuk, ezt jelenti a [{a,c},s] formátum, s végül ezeknek a két elem listáknak a halmazát vesszük.

```
> G := Graph({[{a,b},3],[{a,c},5],[{b,c},2],[{a,e},2],[{b,d},
  4],[{b,f},4],[{c,d},9],[{c,e},3],[{c,f},4],[{d,e},4],[{d,f},
  6],[{e,f},7]});
```

Rajzoljuk föl a gráfot.

```
> DrawGraph(G);
```

Alkalmazzuk az Kruskal algoritmust.

```
> T := KruskalsAlgorithm(G, 'w'):
> `A minimális feszítfa élsúlyainak összege`=w;
```

Ugyanez a Maple beépített eljárásával:

```
> add(GetEdgeWeight(G,u), u=Edges(T));
> `A minimális feszítfa élei a súlyukkal`= map(t->[t,
  GetEdgeWeight(T,t)],Edges(T));
```

Ugyanez a Maple beépített eljárásával:

```
> Edges(T,weights);
```

```
> HighlightSubgraph(G,T,red,green);
> DrawGraph(G);
```
Rajzoljuk föl csak a gazdaságos feszítfát:
```
> DrawGraph(T);
```

## Kézi számítás

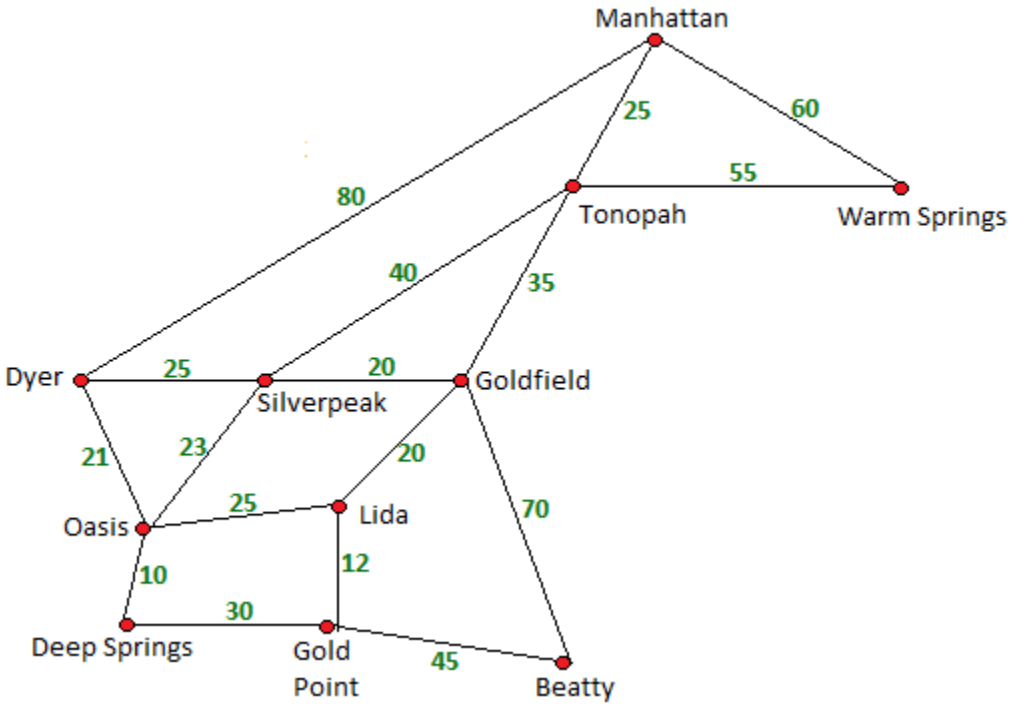
Nem túl nagy méret G gráf esetén papir-ceruzás módszerrel is könnyen meghatározhatjuk a gráf minimális feszítfáját (gazdaságos favázát). Ekkor a következ algoritmus alapján járhatunk el: Törlünk a G-nek körökben szerepl élei közül egy legnagyobb költségt. Az így kapott G1 gráf a G-nek részgráfja; továbbá G1összefügg és G minden pontját tartalmazza. Ezután törlünk a G1 köreiben szerepl egy legnagyobb költség élet, és így tovább. Ha a törlési utasítás már nem hajtható végre, akkor az utóljára kapott F gráfnak már nincs köre, és F a G gráfnak részgráfja. Emellett F összefügg, és a G minden pontját tartalmazza. Bizonyítható, hogy F a G gazdaságos faváza.

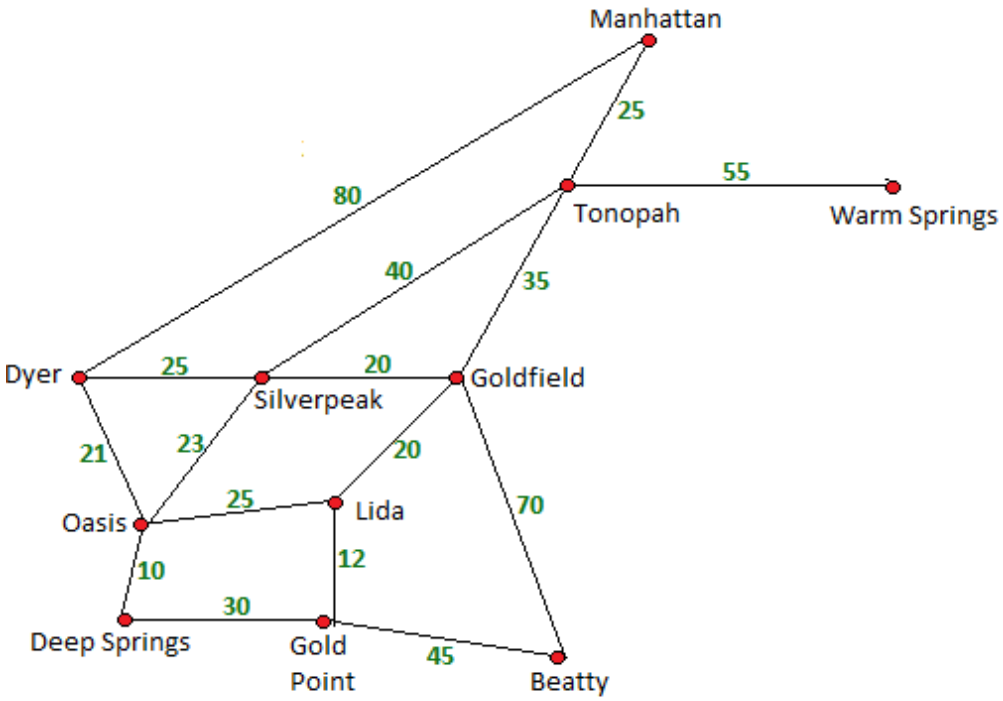
### 8.2.5. Példa

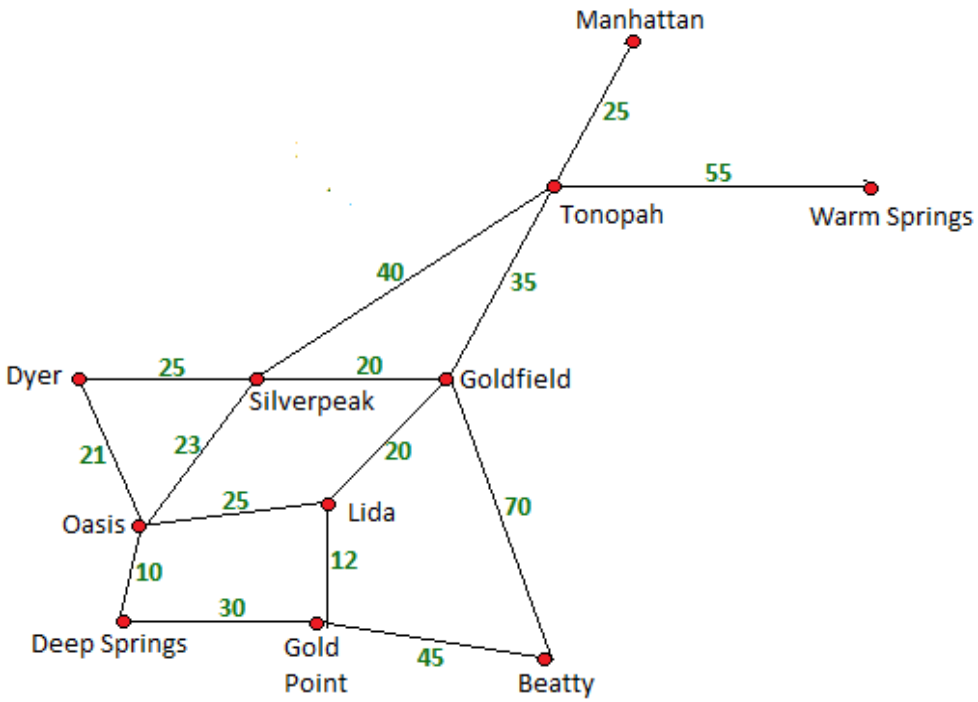
*Bevezet példánk nem túl nagy méret, tehát alkalmas a papir-ceruzás módszerrel történ megoldás bemutatására!*

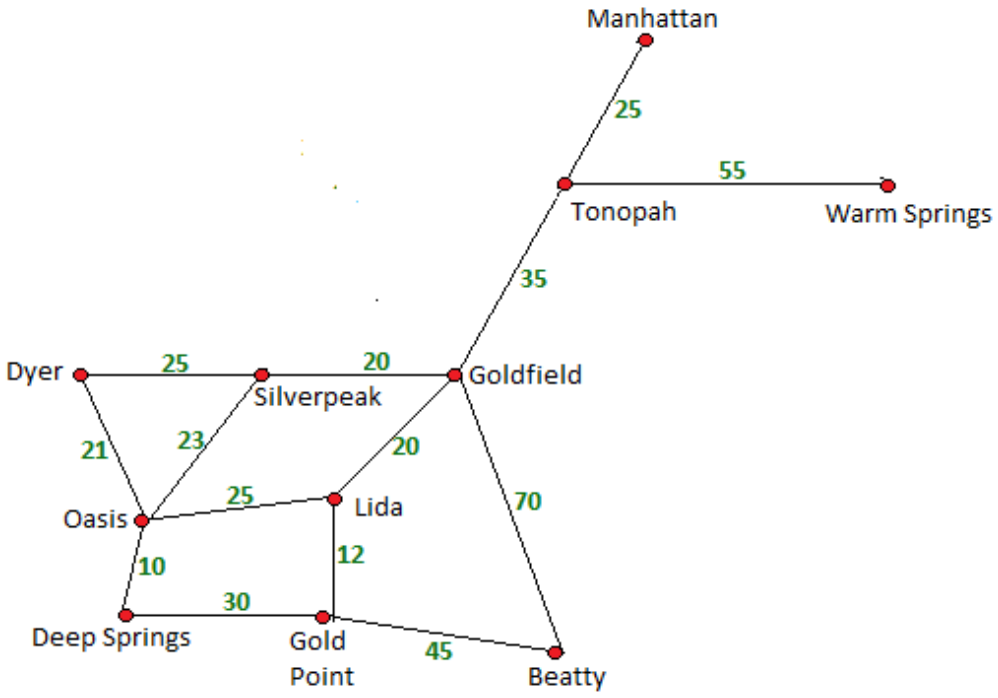
### Megoldás

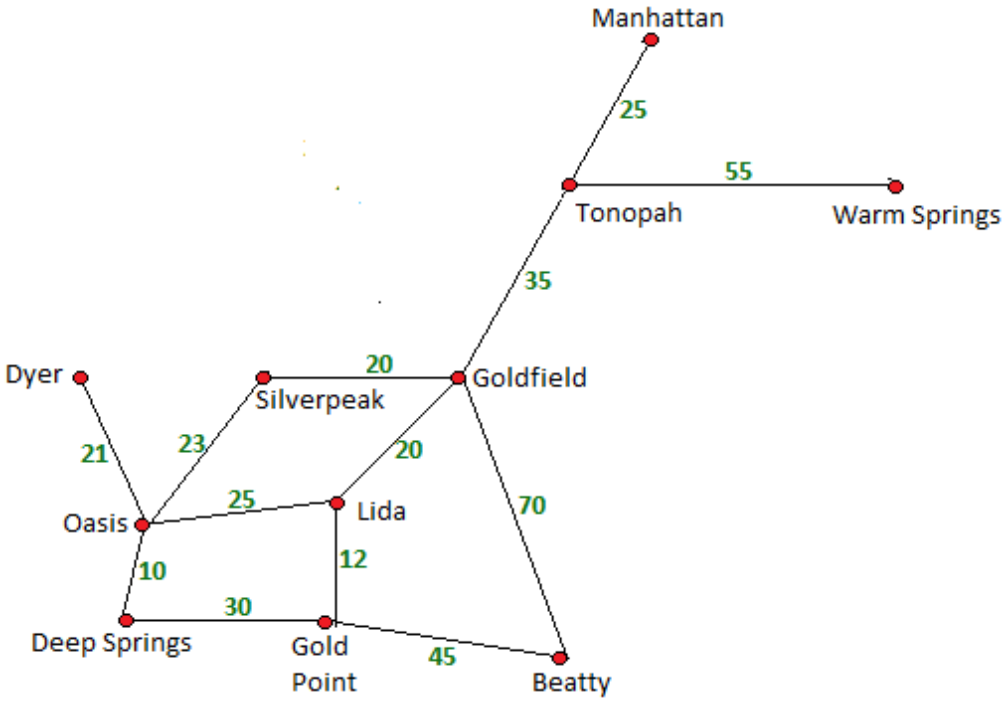
Az esereti gráfot rajzoltuk fel elször, majd sor-folytonosan rendre töröltük a gráf  körökben szerepl élei közül egy legnagyobb költségt. Végül eljutottunk a gazdaságos favázhoz. Ez megegyezik a KruskalsAlgorithm eljárás alkalmazásával nyert favázzal.

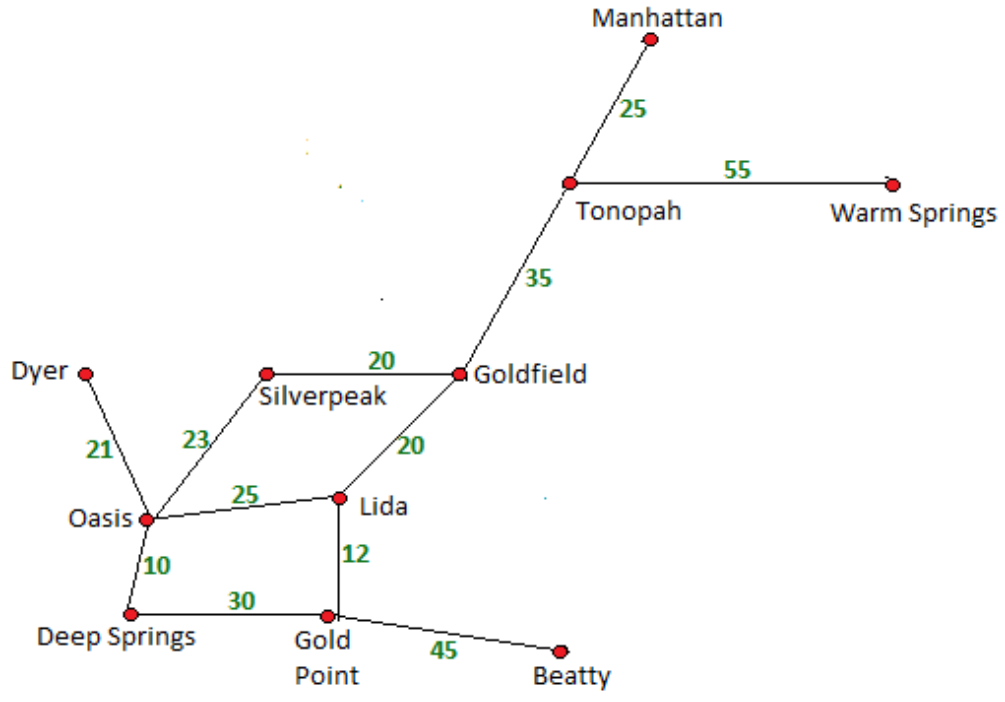

Manhattan
25
60
55
80
Tonopah
Warm Springs
40
35
Dyer
25
20
Goldfield
Silverpeak
21
23
20
25
Oasis
Lida
70
10
12
30
Deep Springs
Gold
Point
45
Beatty

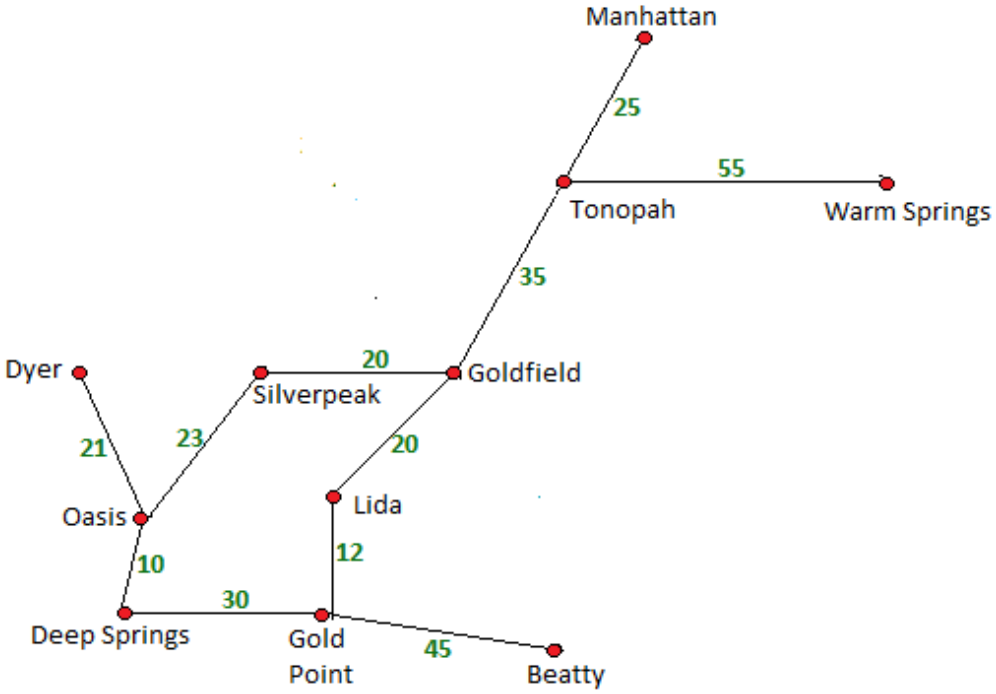

Manhattan
25
55
80
Tonopah
Warm Springs
40
35
Dyer
25
20
Goldfield
Silverpeak
21
23
20
25
Lida
70
Oasis
10
12
30
Deep Springs
Gold
Point
45
Beatty

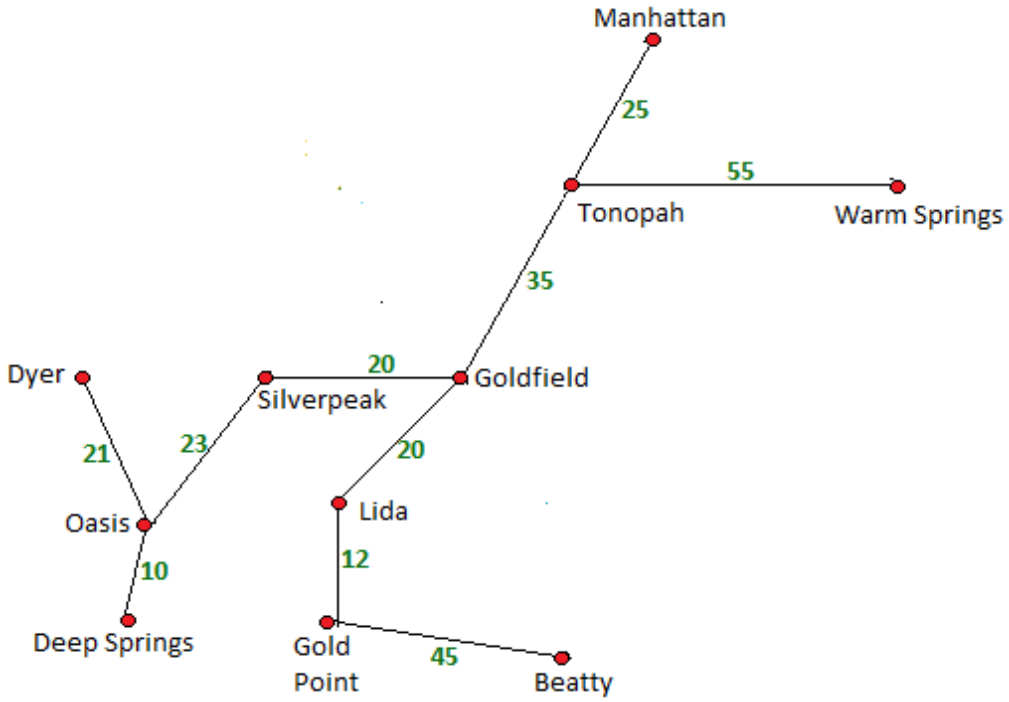

Manhattan
25
55
Tonopah
Warm Springs
40
35
Dyer
25
20
Goldfield
Silverpeak
21
23
20
25
Oasis
Lida
70
10
12
30
Deep Springs
Gold
Point
45
Beatty


Manhattan
25
55
Tonopah
Warm Springs
35
Dyer
25
20
Goldfield
Silverpeak
21
23
20
25
Oasis
Lida
70
10
12
30
Deep Springs
Gold
Point
45
Beatty


Manhattan
25
55
Tonopah
Warm Springs
35
Dyer
20
Goldfield
Silverpeak
21
23
20
25
Oasis
Lida
70
10
12
30
Deep Springs
Gold
Point
45
Beatty


Manhattan
25
55
Tonopah
Warm Springs
35
Dyer
20
Goldfield
Silverpeak
21
23
20
25
Oasis
Lida
10
12
30
Deep Springs
Gold
Point
45
Beatty


Manhattan
25
55
Tonopah
Warm Springs
35
Dyer
20
Silverpeak
Goldfield
21
23
20
Oasis
Lida
10
12
30
Deep Springs
Gold
Point
45
Beatty




# A legrövidebb út, Dijkstra algoritmusa

## Bevezetés

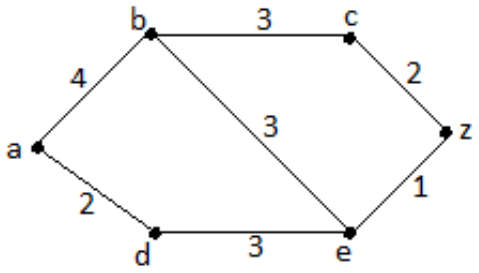
*A boldogsághoz vezet legrövidebb út: az önismeret. (Hioszi Tatiosz)*

Gyakori feladat, hogy két pont között meg kell találnunk a legrövidebb utat.A legrövidebb szó itt sok mindent jelenthet: jelentheti azt, hogy az út hossza legyen minimális; gondolhatunk arra, hogy az utazás ideje legyen a legrövidebb; vagy az is elképzelhet, hogy olyan utvonalat keresünk, amelynek a költsége minimális. Akármelyik esetrl is legyen szó, a probléma mndig modellezhet gráfokkal a következképpen:

Legyen adva egy irányítás nélküli, súlyozott él G gráf, amelyben minden él súlya ( költsége) pozitív. Tekintsük továbbá a gráf két pontját, ***a***-t és ***z***-t. Keressük meg az a-ból z-be vezet legkisebb költség utat, vagy más kifejezéssel a legrövidebb utat. Két pont közötti út költsége alatt az útat alkotó élek költségének  összegét értjük.
A Dijkstra-algoritmus ezt a feladatot oldja meg. Az algoritmust Edsger Wybe Dijkstra holland informatikus 1956-ban alkotta meg, és 1959-ben publikálta.

### 8.3.1. Példa

*Mennyi az **a**-ból **z**-be vezet legrövidebb, más kifejezéssel élve kigkisebb költség út hossza? Adjuk is meg ezt az utat!*




**Megoldás**

Bár ilyen kis méret gráf esetében szemlélet alapján is meghatározható a legkisebb költség, vagy másképp szólva a legrövidebb út, mi most nem így teszünk. A következkben részletesen kifejtett módszert próbáljuk elkészíteni. Két olyan út van, amely (a terminális csúcs eléréséig) nem tartalmaz a-n kívül csúcsot, a,b és a,d. Az a,b illetve a,d út hossza rendre 4 ill. 2, ezért d az a-hoz legközelebbi csúcs. Ezután megkereshetjük a következ legközelebbi csúcsot az összes olyan utat tekintetbe véve, amely csak az a-n és a d-n haladhat át, mieltt az utolsó csúcsba érkeznék. A legrövidebb ilyen út b csúcsba még most is a,b, s ez 4 hosszúságú, az e csúcsba vezet legrövidebb út a,d,e, és ennek hossza 5. Tehát a következ legközelebbi csúcs a-hoz b. A harmadik a-hoz legközelebbi csúcs megkereséséhez az összes olyan utat kell megvizsgálnunk amely csak az a,b és a d csúcson halad át, mieltt a "végállomásra" érne. Van egy 7 hosszúságú út c-ig, az a,b,c és egy 6 hosszúságú z-ig, az a,d,e,z út. Ezért z a következ legközelebbi csúcs az a-hoz és a legrövidebb út hossza 6.

## A Dijkstra algoritmus leírása

Az elz szekció példája illusztrálja a Dijkstra algoritmust. Természetesen, ahogy említettük már, ebben az esetben eredményesek lehettünk volna az összes lehetség átvizsgálásával is. De ilyen módon nagyobb méret gráfoknál már nem lehetünk sikeresek. A következkben a probléma általános tárgyalásával foglalkozunk, azaz a legrövidebb út megkeresésével tetszleges, irányítás nélküli, súlyozott, egyszer gráfok esetében. A Dijkstra algoritmus úgy halad elre, hogy megkeresi az a-tól a szomszédos csúcsig a legrövidebb utat s annak a hosszát hosszát, majd a legrövidebb utat a-tól a második csúcsig, és így tovább, míg végül megtalálja az a-tól a z-ig tartó legrövidebb utat.
Az algoritmus iterációk sorozatával valósul meg. A csúcsok kitüntetett halmaza konstruálódik iterációs lépésenként egy-egy új csúcs hozzáadásával. Az iteráció minden egyes lépése egy címkézési eljárást valósít meg. Ebben az eljárásban egy w csúcs címkéje a legrövidebb olyan út hossza a-tól w-ig, amely út csak olyan csúcsokat tartalmaz, amelyek már a megkülönböztetett halmazhoz tartoznak. A megkülönböztetett halmazhoz hozzáadott csúcs minden egyes lépés során egy minimális címkéj csúcs azok közül, amelyek nem tartoznak a megkülönböztetett halmazhoz.

Ezek után részletesen leírjuk a Dijkstra - algoritmust. Els lépésként az *a* csúcs 0, a többi csúcs ∞ címkét kap. Ezt így jelöljük:
$L_0(a) = 0$ és $L_0(v) = \infty$.A 0 index azt mutatja, hogy még nem történt iteráció, a 0-adik iterációs lépésnél tartunk.A címkék a legrövidebb útat mutatják az a csúcstól a v csúcsig. Most még értelemszerűen az út egyeteln csúcsból áll csak.(Ha egy csúcshoz nincs út az a-tól, akkor ∞ a legrövidebb út hossza.)
Az algoritmus a csúcsok megkülönböztetett halmazának formálásával halad elre. Jelölje $S_k$ ezt a halmazt a címkézési eljárás k-adik iterációs lépése után. Az eljárást $S_0 = \varnothing$-val kezdjük. $S_k$ úgy képzdik az $S_{k\_1}$-bl, hogy hozzáveszünk a csúcsokhoz egy minimális index, az $S_{k\_1}$-be nem

tartozó, u csúcsot. Miután u-t hozzávettük az $S_k$-hoz, felülírjuk az összes, $S_k$-hoz nem tartozó csúcs indexét, oly módon hogy $L_k(v)$, a v csúcs k-adik iterációs lépés utáni címkéje a legrövidebb olyan út hossza az a-tól a v-ig, amely csak $S_k$-hoz tartozó csúcsokat tartalmaz (azaz olyan csúcsokat, amelyek u-val együtt már a megkülönböztetett halmazban vannak.)
Legyen v egy, nem az $S_k$-ban lev csúcs.A v címkéjét frissítend, jegyezzük meg, hogy $L_{\mathrm{k}}(v)$a legrövidebb olyan út az a-tól a v-ig, amely csak $S_k$-beli csúcsokat tartalmaz. A frissítést hatékonyan végezhetjük, ha felhasználjuk a következ megfigyelést: A legrövidebb olyan út, amely csak $S_k$ elemeit tartalmazza, kétféleképp jöhet létre:

- a legrövidebb út a-tól z-ig, amely csak $S_{k-1}$elemeit tartalmazza (azaz a megkülönböztetett csúcsok az u nélkül).
- vagy a $k-1$-edik lépés során létrejöv, az a-tól az u-ig vezet út, hozzáadva az $(u, v)$ élet.

Formulával kifejezve:

$$L_{\mathrm{k}}(a, v) = \min\{L_{\mathrm{k}-1}(a, v), L_{\mathrm{k}-1}(a, u) + w(u, v)\}$$

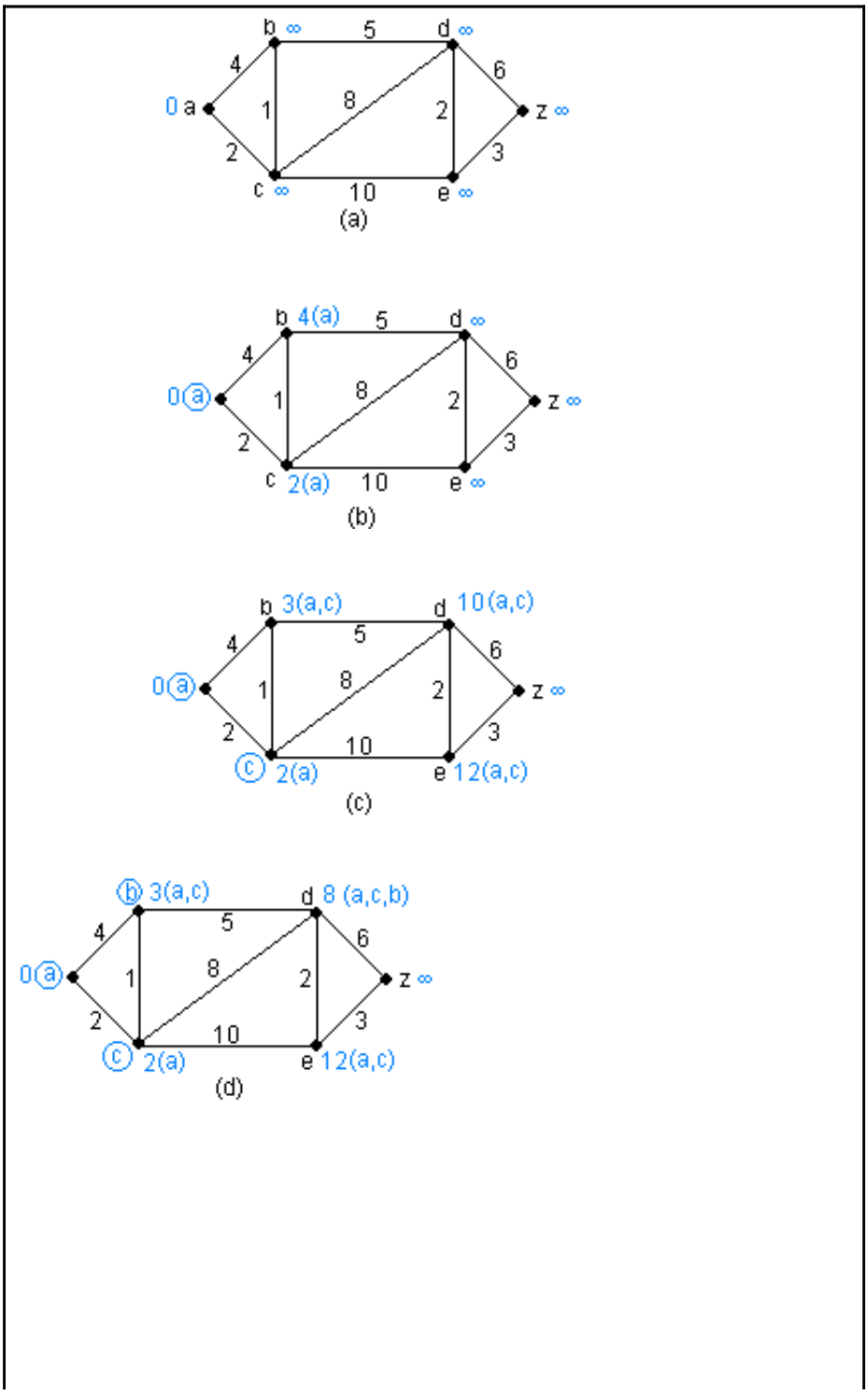
Az eljárás iteratív, egymás után adódnak hozzá a csúcsok a megkülönböztetett halmazhoz, utóljára a *z* csúcs. Amikor *z* is hozzáadódott a megkülönböztetett halmazhoz, akkor címkéje a legrövidebb út az *a* csúcstól a *z* csúcsig.

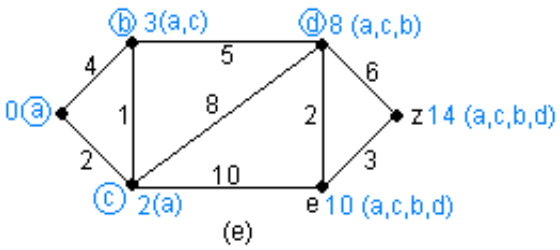
## Pszeudokód

```
procedure Dijkstra(G:súlyozott, összefügg,
egyszer gráf, pozitív súlyokkal)
{G csúcsai a = v0, v1, . . . , vn = z, az élek súlyai
w(vi, vj), ahol w(vi, vj) =∞, ha {vi, vj}nem éle G-
nek}
for i:=1 to n
  L(vi) := ∞
  L(a):=0
S:=∅
{a címkéket úgy inicializáltuk, hogy az a csúcs
címkéje 0, minden más címke ∞, és S üres
halmaz.}
while z∉S
begin
  u : = egy csúcs, amely nincs S-ben, s amelyre L
(u) minimális
  S : = S∪{u}
  for minden v csúcsra, amely nincs S-ben
    if L(u) + w(u, v) < L(v) then
L(v) := L(u) + w(u, v)
  {ezáltal hozzáadódik egy  minimális címkéj
csúcs S-hez és minden nem S-beli csúcs címkéje
megújul}
end {L(z) = az a-ból z-be vezet legrövidebb út}
```

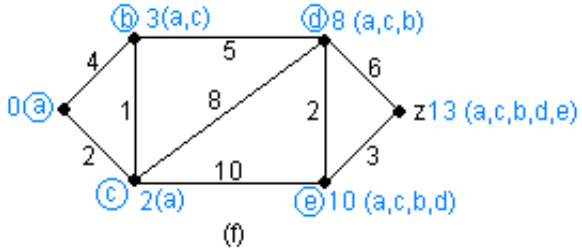
## Példa

Az elzk illusztrációjaként használjuk a Dijkstra-algoritmust az (a) ábrán látható súlyozott gráf ***a*** *és* **z** csúcsa közötti legrövidebb út megkeresésére.
Az eljárás lépéseit az alábbi ábrasorozaton jelenítettük meg. Az iteráció minden egyes lépésénél az $S_k$ -hoz tartozó csúcsokat bekarikáztuk és egyes lépésénél megadtuk az a csúcstól az $S_k$-ba tartozó csúcsokig vezet legrövidebb útat. Az algoritmus a *z* csúcs bekarikázásával ér véget.
Látható, hogy a-tól z-ig a legrövidebb út az a,c,b,d,e,z, és ennek hossza 13.

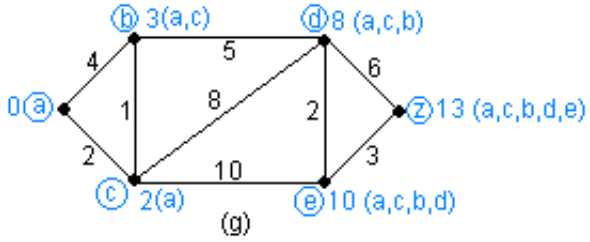




(e)



(f)



(g)

## Maple eljárás

Oldjuk meg az elz szekcióban szerepl feladatot  a Maple Graphtheory csomag DijkstrasAlgorithm eljárásával is!

```
> restart:
> with(GraphTheory):
```

Tekintsük a G összefügg, egyszer, súlyozott gráfot:

```
> V:=[a,b,c,d,e,z];
> E:={[{a,b},4],[{a,c},2],[{b,c},1],[{b,d},5],[{c,d},8],[{c,e},
  10],[{d,e},2],[{d,z},6],[{e,z},3]};
> G:=Graph(V,E);
```

Rajzoljuk föl a gráfot!

```
> DrawGraph(G);
```

Keressük meg az 1-es csúcsból a 8-as csúcsba vezet legrövidebb útat!

```
> DijkstrasAlgorithm(G, a, z);
```

Az algoritmussal az elz megoldással megegyezen megkaptuk, hogy a legrövidebb út: a, c, b, d, e, z, és ennek hossza 13.

Az eljárás outputja két elem lista. Az els elem a lerövidebb útat adó csúcsok listája, a második elem az út költsége.

```
> Csucsok:=op(1,%);
```

A legrövidebb útat az ábrán kiemelhetjük: a csúcsokat lila, az éleket piros színnel jeölve:

```
> HighlightVertex(G,Csucsok,magenta);
> HighlightEdges(G,Trail(op(Csucsok)),red);
> DrawGraph(G,style=circle);
>
```

## Ellenrz kérdések

1. Mit értünk a gyökeres fa preorder, indoorer, postorder bejárása alatt?
2. Értelmezze a kifejezés prefix, infix és postfix kódját!
3. Definiálja a faváz (feszít fa) fogalmát!
4. Mit értünk miniális érték faváz alatt?
5. Melyek a Kruskal algoritmus lépései?
6. Milyen feladatot old meg Dijkstra algoritmusa?